UNLIMITED

(2)

Report No. 91019

Report No. 91019

ROYAL SIGNALS AND RADAR ESTABLISHMENT, MALVERN

SECURE REFINEMENT

Author: Paul Smith†

† Secure Information Systems Ltd
Sentinel House, Harvest Crescent
Ancells Park, Fleet
Hampshire, GU13 8UZ

PROCUREMENT EXECUTIVE, MINISTRY OF DEFENCE
RSRE
Malvern, Worcestershire.

February 1991

CONDITIONS OF RELEASE

0105450 303826

.....

DRIC U

.....

DRIC Y

Reports quoted are not necessarily available to members of the public or to commercial organisations.

DEFENCE RESEARCH AGENCY
ELECTRONICS DIVISION

REPORT 91019

Title: Secure Refinement

Author: Paul Smith†

Date: 12th February 1991

Abstract

Behavioural equivalence is identified as a mathematical property which captures the notion of secure refinement.

This report has been furnished for use by Her Majesty's Government under the terms and conditions of contract SLS42c/719 from the Defence Research Agency Electronics Division

† Secure Information Systems Limited
Sentinel House, Harvest Crescent, Ancells Park, Fleet, Hampshire, GU13 8UZ
Telephone (0252) 811818
Fax. (0252) 811435

INTENTIONALLY BLANK

Contents

INTENTIONALLY BLANK

1 Introduction

The confidentiality of information in multi-level secure systems is maintained by sanitising requests for information according to the requester's clearance. However, information can flow indirectly through subtle modulations of shared resources — so-called "signalling channels". These channels can be avoided in specifying secure systems, but as a system specification is progressively refined in the design process, extraneous behaviours can easily be introduced, opening up more and more signalling channels. The process of constructing a system which is free from signalling channels is therefore two-fold: firstly, show that the system specification is free from threatening signalling channels, by demonstrating its correspondence to a security policy model; secondly, prove that successive design specifications do not introduce additional signalling channels, i e. they are 'secure refinements' of the system specification.

This paper identifies "behavioural equivalence" as a mathematical notion of secure refinement for SMITE-based systems, and proposes suitable proof obligations It has been suggested that a SMITE implementation is secure if its specification is Terry-Wiseman secure [Terry89], and its safety-refined implementation is isomorphic to the specification. Given that implementations are protected by an information hiding mechanism [Wiseman90], it would seem that isomorphic implementations would be too restrictive - they are certainly artificially complex.

The requisite mathematical notions are recalled in section 2 A discussion of secure refinement in section 3 motivates the choice of 'behavioural equivalence' as a useful definition The definition is investigated in terms of model-oriented Z specifications [Spivey89] in section 4, and a proof obligation for behavioural equivalence is posited, behavioural equivalence is difficult to characterise for non-deterministic systems, and the limitations are sketched In section 5, behavioural equivalence is formally defined as a means to proving the validity of the proof obligation

2 Signatures, Algebras and Morphisms

Z specifications describe algebras, but the algebraic concepts underlying notions of correctness are best understood in the context of algebraic specifications The main definitions are recalled in this section.

Informally, a signature Σ is a set of *sort* names, together with a set of *operator* names each of which has a declared syntactic structure (cf. type) formed from $\times$ and $\rightarrow$. For example, the signature of a cut-down algebra of number sets, called *Bunch*, might be written.

sorts	bunch, bool, nat
opns	empty : bunch
	insert: bunch $\times$ nat $\rightarrow$ bunch
	isin : bunch $\times$ nat $\rightarrow$ bool

A *Σ-Algebra* is a signature, Σ, together with an assignment mapping each sort to a set of values, called its *carrier*, and each operator to a total function, called an *operation*[1]

[1]In practice, the operations would not be defined directly in this manner Rather, the operations would be defined indirectly by equations on the operators The algebra which assigns total functions and carrier sets to each operator and sort is then chosen from the set of all algebras satisfying the

The carrier of sort s in algebra A is written $|\ A_s\ |$. For example, let the carriers of a *Bunch*-algebra A be the familiar mathematical sets of numbers and binary logic values, thus:

$$
\begin{aligned}
|\ A_{bunch}\ | &= \mathbb{PN} \\
|\ A_{bool}\ | &= \{0, 1\} \\
|\ A_{nat}\ | &= \mathbb{N}
\end{aligned}
$$

The operations are the familiar ones from set theory.

$$
\begin{aligned}
A_{empty} &\triangleq \{\} \\
A_{insert}(b,n) &\triangleq b \cup \{n\} \\
A_{isin}(b,n) &\triangleq \text{if } n \in b, 0 \text{ otherwise}
\end{aligned}
$$

This readily accords with our expectations, and could be offered as a specification of a system of bunches. The *Bunch*-algebra B, on the other hand, chooses unordered sequences to represent bunch, thus

$$
\begin{aligned}
|\ B_{bunch}\ | &= \text{seq}\,\mathbb{N} \\
|\ B_{bool}\ | &= \{0, 1\} \\
|\ B_{nat}\ | &= \mathbb{N}
\end{aligned}
$$

with the operations

$$
\begin{aligned}
B_{empty} &\triangleq \langle\rangle \\
B_{insert}(b,n) &\triangleq b \frown \langle n \rangle \\
B_{isin}(b,n) &\triangleq \text{if } n \in \text{ran } b, 0 \text{ otherwise}
\end{aligned}
$$

Now if A is the specification, is B a correct implementation? This is precisely the safety refinement problem. B is a safety refinement of A if B is *homomorphic* to A [Jones86].

A *homomorphism* between two Σ-algebras is a total surjection between the carrier sets of the two algebras which preserves the semantics of the operations. To show that an algebra B is homomorphic to algebra A entails

1. constructing an interpretation of the carriers of B (wrt the carriers of A) i.e. a total surjection from each carrier set of B to its corresponding carrier set of A (cf *retrieve functions* [Jones86]),

2. for each operation, and each input: proving that B's definition interprets to A's definition.

In the example, suitable interpretations of the carriers of B in terms of the carriers of A is

$$
\begin{aligned}
\text{ibunch} &= \lambda\, b : \text{seq}\,\mathbb{N} \cdot \text{ran } b \\
\text{ibool} &= \{1 \mapsto 1, 0 \mapsto 0\} \\
\text{inat} &= \text{id}\,\mathbb{N}
\end{aligned}
$$

To prove the *insert* operation, for example, we must show for any bunch b and any

---

equations, depending on how the equations are interpreted (e.g. 'term algebra' [Goguen78]). The algebras are explicitly defined in this section purely to motivate the algebraic notions underlying refinement.

number n,

$$ibunch(B_{insert}(b,n)) = A_{insert}(ibunch(b), inat(n))$$

which is expanded successively using the definitions of B_{insert} and A_{insert}, then *ibunch* and *inat*, as follows:

$$\Rightarrow \quad ibunch(b \frown \langle n \rangle) = (ibunch\ b) \cup \{inat\ n\}$$
$$\Rightarrow \quad ran\ (b \frown \langle n \rangle) = (ran\ b) \cup \{n\}$$

which is a property of the range of a catenated sequence. It is easy to show that the other operations preserve the interpretation, and thus B is homomorphic to A

Two algebras are *isomorphic* if they are homomorphic as above, and the interpretation is a total bijection. In the example, the only algebra (with $| B_{bunch} | = seq\ \mathbb{N}$) which fits this property uses ordered injective sequences For example, the set *{1,3,5}* must be represented by the sequence *<1,3,5>*, and cannot be represented by *<1,5,3>* for instance. Also excluded are representations which include 'junk' - e g *<1,3,3,3,5>* A more telling example involving junk is a sequence and pointer, for instance:

<1, 3, 5, 9>
↑

The fourth element of this sequence is junk, but its presence renders the representation non-isomorphic. The representation could be made isomorphic by insisting that the number of junk elements is constant, and that all junk is the same value (e g 0) Such a compromise can be a nuisance when scaled up (e g. field padding in database)

3 Secure Implementations

Algebra B above is not isomorphic to A, because the ordering and duplication of elements within a sequence is not unique One might argue that B is an insecure implementation of A, given the knowledge that elements are catenated successively by *insert* However, by hiding the carrier set of *bunch* from every user of the algebra B, any information subtly encoded within the stored sequence cannot be decoded - information can be transmitted, but never received[1]. This is the purpose of data hiding in SMITE. Under this interpretation, we have shown a non-isomorphic, secure implementation

Other non-isomorphic implementations might not be secure, however For example, suppose the signature in the above example had another sort, called *display*, and an operation

$$show : bunch \rightarrow display$$

which shows the user which elements are in the *bunch* Suppose the algebra A defines the new carrier and operation as

$$| A_{display} | = \mathbb{FN}$$
$$A_{show}(b) \triangleq b$$

but suppose B defines it as

[1]Other than by observing modulations in response time

$$
\begin{aligned}
&| B_{display} | = \text{seq}\,\mathbb{N} \\
&B_{show}(b) \triangleq b
\end{aligned}
$$

The problem is that the user of B can now observe modulations in the use of the bunch by performing *Show*. B is not isomorphic to A Clearly, 'hiding' display would contradict its purpose, so we must reject B as an insecure implementation of A. Even if the display was always presented in the same order, duplicate elements would constitute a signalling medium Therefore, suppose algebra C is the same as B except that it does not store duplicate elements (the stored sequences are injective), and C_{show} displays the stored sequence in a fixed order, thus:

$$
\begin{aligned}
&| C_{bunch} | = \text{iseq}\,\mathbb{N} \\
&| C_{display} | = \text{iseq}\,\mathbb{N} \\
&C_{show}(b) \triangleq \text{order}\, b
\end{aligned}
$$

where the function *order* re-orders any sequence of numbers in non-decreasing order (e g *order* $\langle 3,5,1\rangle = \langle 1,3,5\rangle$) C is a secure implementation, but it is still not isomorphic to A[1], because any set of numbers corresponds to possibly many distinct re-orderings of the stored sequence

What then is a secure implementation? The answer is one which *behaves* the same as the specification with respect to the non hidden sorts Hence *Behavioural Equivalence* [Sannella83] The following definition of b e is given in [Sannella84]

> "Two algebras are behaviourally equivalent with respect to a set of observable sorts if and only if they give corresponding answers to every computation taking inputs of observable sorts and yielding a result of observable sort "

In the equational specifications expounded in [Sannella83,84, Goguen78], this property can be shown by checking the definitions of observable operations against the equations on observable sorts Their only problem is finding an algebra which corresponds to a sufficiently loose interpretation of the equations (e g 'initial' algebra [Goguen78])

Z specifications are usually not described exclusively by equations Instead we define pairs of models, one of which is the specification, the other the implementation, and we don't have equations to test for behavioural equivalence Also, Z specifications are non-deterministic in general, so they describe sets of algebras We need to show that representative algebras of each specification are behaviourally equivalent, without reference to any equations This might be achieved by showing a one-to-one correspondence between output variables in operation schemas This is investigated informally in the next section

4 Behavioural Equivalence in Z

4.1 Z Specifications and Refinements

The *Bunch* example from above is re specified in Z Then a proof obligation for behavioural equivalence is posited and investigated

The carrier set of the bunch in algebra A above is represented by a state schema in Z, as

[1] Although the interpretation for display is bijective

follows:

$$Bunch \triangleq [b : \mathbb{P}\mathbb{N}]$$

The operations are also defined using schemas in the usual manner, one schema per operator of *Bunch*.

$$\begin{aligned} Empty &\triangleq [Bunch' \mid b' = \{\}] \\ Insert &\triangleq [Bunch; Bunch'; i? : \mathbb{N} \mid b' = b \cup \{i?\}] \\ display &== \mathbb{P}\mathbb{N} \\ Show &\triangleq [Bunch, d! : display \mid d! = b] \end{aligned}$$

The (insecure) bunch representation (algebra *B* above) is represented by the following schemas:

$$\begin{aligned} Bunch_I &\triangleq [b_I : seq\,\mathbb{N}] \\ Empty_I &\triangleq [Bunch_I' \mid b_I' = \langle\rangle] \\ Insert_I &\triangleq [Bunch_I; Bunch_I'; i? : \mathbb{N} \mid b_I' = b_I \frown \langle i? \rangle] \\ display_I &== seq\,\mathbb{N} \\ Show_I &\triangleq [Bunch_I; d_I' : display_I \mid d_I' = b_I] \end{aligned}$$

The relationship between the states of these two specifications is described by the following abstraction schema:

$$Abs = [Bunch; Bunch_I \mid b = ran\, b_I]$$

This schema corresponds to the homomorphic mapping *ibunch* above

The (secure) implementation algebra *C*, differs in the treatment of injective sequences in the state and the ordering of the display

$$\begin{aligned} Bunch1_I &\triangleq [b_I : iseq\,\mathbb{N}] \\ Insert1_I &\triangleq [Bunch_I, Bunch_I' : \mathbb{N} \mid \\ &\qquad i? \notin ran\, b_I \wedge b_I' = b_I \frown \langle i? \rangle \vee \\ &\qquad i? \in ran\, b_I \wedge b_I' = b_I] \\ display1_I &== \{s : iseq\,\mathbb{N} \mid s \in ran\, order\} \\ Show1_I &\triangleq [Bunch_I; d_I' : display_I \mid d_I' = order\, b_I] \end{aligned}$$

In the example, neither of the implementations is isomorphic, because many distinct re-orderings of the sequences correspond to the same set of numbers (many-to one) In order to make the last implementation isomorphic, we would need to store the sequence in sorted order, rather than ensuring that only ordered sequences are output This seems convenient enough in this simple example, but it might not be so convenient in a more complex system On the other hand, some systems can only be implemented efficiently with unique stored representations. The point is that the implementer should be free to choose the most convenient representation, without unnecessary algebraic constraints

This leads us to consider a weaker notion of secure refinement, which is independent of the algebraic relationship between specifications (morphisms) A refinement is secure if it is correct, and it does not introduce new observable behaviours. Therefore, we only need to check that corresponding operations behave equivalently up to their outputs

4.2 Proving Behavioural Equivalence

The notion of 'behavioural equivalence' has been proposed for some years as a semantics for abstract data types [Sannella84], where a data type description encapsulates its private state within a fixed set of operations. The operations are permitted to manipulate the state, but otherwise, the operations must be used to effect the state indirectly. The operations make observations on the state by proxy, but cast those observations in terms of other, user-visible data types Once an abstract data type has been specified, it does not matter how the implementation works internally, as long as it offers the same observable behaviour promised by the specification.

The attraction of behavioural equivalence in the interpretation of abstract data types is that it more closely reflects the 'black box' view of an abstract data type. Morphisms, on the other hand, deal with the internal structure of the data types - more akin to a 'white box' view The data hiding mechanism in SMITE is based on the black box approach. Black-box interpretation is a weaker notion of 'correctness' in the following sense. for any algebra, the set of behaviourally equivalent algebras is larger that the set of 'morphous algebras In software engineering terms, the implementer is less constrained. The treatment of behavioural equivalence in abstract data types has focussed on algebraic specifications, but Z specifications are model-based A definition of behavioural equivalence in terms of Z specifications follows

In general, two systems are 'behaviourally equivalent' if there is a one-to-one correspondence between the results of 'executing' corresponding sequences of operations Inductively, this is the case if corresponding operations have a one-to one (actually 'bijective') output correspondence Consider state spaces S and S_i, related by the abstraction Abs, along with the operation OP and its implementation OP_i, as follows.

$$
\begin{array}{lcl}
Abs & \widehat{=} & [\ S, S_i \mid \ldots\ ] \\
OP & \widehat{=} & [\ S; S'; \iota?.I, o! O \mid \quad] \\
OP_i & \widehat{=} & [\ S_i; S_i'; \iota? I, o_i! O_i \mid \ldots\ ]
\end{array}
$$

(Assume that inputs are not refined)

For corresponding operations OP and OP_i, the *output correspondence* is the following relation

$$out == \{ OP, OP_i; Abs; Abs' \cdot (o_i!, o!) \}$$

We must show that the output correspondence is a total bijection[1], thus

$$\vdash out \in O_i \rightarrowtail O$$

The obvious way of proving this property is to construct the output correspondence relation from the operation definitions and the abstraction, and show it is bijective Indeed, such an output correspondence relation is required in order to prove the 'correctness' of the operation[2] In the case of the *Show* operation in the first (insecure) refinement above, the relationship is as follows

$$\{ Show, Show_i; Abs, Abs' \cdot (o_i!, o!) \}$$

[1] In practice, the output correspondence will often be an identity function
[2] Although the published refinement methods usually ignore this

which can easily be constructed from the *Show* and $Show_t$ operations where $o_t' = order\ b_t$ and $o' = b$, thus:

$$\{b_t : \text{seq}_1 \mathbb{N} ; b : \mathbb{F}\mathbb{N} \mid (b_t, b)\}$$

but the abstraction *Abs* equates $b = ran\ b_t$, so the output correspondence reduces to

$$\{b_t : \text{seq}_1 \mathbb{N} ; b : \mathbb{F}\mathbb{N} \mid (b_t, ran\ b_t)\}$$

which, although functional, is clearly not bijective, since some distinct sequences have the same range (e.g. *ran* <2,3> = *ran* <3,2>). The refinement is therefore not secure. The correspondence for a second (secure) refinement is

$$absout1 = \{Show; Show1_t; Abs; Abs' \mid (o_t', o')\}$$

which, since $o_t' = order\ b_t$, and $o' = b$ in the definitions of *Show* and $Show1_t$, is as follows:

$$\{b_t : \text{seq}_1 \mathbb{N} ; b : \mathbb{F}\mathbb{N} \mid (order\ b_t, b)\}$$

but again the abstraction reduces the output correspondence to

$$\{b_t : \text{seq}_1 \mathbb{N} ; b : \mathbb{F}\mathbb{N} \mid (order\ b_t, ran\ b_t)\}$$

i.e.

$$\lambda\, b_t : display1_t \cdot ran\ b_t$$

which is a total bijection because each set of numbers has a unique order. The second refinement, with its non-duplicate, unordered sequence is therefore secure.

4.3 Non-determinism

Behavioural equivalence is only valid for deterministic specifications[1], where the outputs of operations are uniquely determined. Otherwise, non-determinism renders the output correspondence non-functional. For example, suppose the bunch specification provided an operation to select an arbitrary number, thus

$$Select \triangleq [\ Bunch; n! : \mathbb{N} \mid b \neq \{\} \wedge n! \in b\ ]$$

The number of possible outcomes from Select is $\#b$. Suppose its implementation always (i.e. deterministically) selects the minimum number, i.e.

$$Select_t \triangleq [\ Bunch_t, n_t! : \mathbb{N} \mid b_t \neq \{\} \wedge n_t! = 1 \text{st}(order\ b_t)\ ]$$

Now the output correspondence is as follows

$$\{b_t : \text{seq}_1 \mathbb{N} ; n : ran\ b_t \mid (head(order\ b_t), n)\}$$

which is a (one-to-many) relation since, for example if $b_t = \langle 7,2 \rangle$, the correspondence is

$$\{2 \mapsto 7, 2 \mapsto 2\}$$

[1]The author is grateful to Professor C B Jones for pointing this out.

If the implementation is also non-deterministic, the output correspondence is a many-to-many relation.

Although the problem is raised by non-deterministic specifications, it is the systems which must be deterministic, in other words the specification may be 'loose', provided only deterministic interpretations of the specification are intended. Our definition of secure refinement is based on properties of the specification, not a system as such, so we are forced to assume that the specification is itself deterministic (with respect to outputs). This raises a further proof obligation for the operations.

The restriction to deterministic specifications does not preclude 'looseness' in the definition of constants [Spivey88]. For example, our selection operation could choose its value in a fixed (i.e. functional) manner, but the basis of the choice can be left unspecified, thus:

$$choose : \mathbb{P}_1 \mathbb{N} \rightarrow \mathbb{N}$$
$$Select \mathrel{\widehat{=}} [\ Bunch; n! : \mathbb{N} \mid b \neq \{\} \wedge n! = choose\, b\ ]$$

The above implementation specification, $Select_1$, is only correct if the choice of the minimum stored number is in one-to-one correspondence with *choose*, and it is consistent with *choose*.

The specification of state variables can be non-deterministic, as long as the non-determinism is resolved by the specification of output variables. In an abstract specification, i.e. without redundancy etc., non-determinism in the state is likely to give rise to non-determinism in the outputs. Implementation specifications, on the other hand, introduce genuine redundancy, so non-deterministic state transformations are more likely to be concealed. In the bunch example, the secure implementation determined the order of the bunch sequence from the order of insertions. This was sorted for output. However, the implementation specification for *Insert* could have made a non-deterministic choice for the position of the new number in the state, thus:

$$Insert1_1 \mathrel{\widehat{=}} [\ Bunch_1; Bunch_1'; i? : \mathbb{N} \mid$$
$$i? \notin ran\, b_1 \wedge ran\, b_1' = ran\, b_1 \cup \{i?\} \vee$$
$$i? \in ran\, b_1 \wedge b_1' = b_1\ ]$$

This even allows existing numbers to be re-ordered The important point is that the system never outputs any clues concerning the stored order. Notwithstanding non-determinism in the specification of state transformations, it will often be more convenient and efficient to factor output filtering to the state itself. In terms of the example, the bunch would be stored in number order. This makes the implementation isomorphic to the specification, purely as a matter of convenience. In practice, other redundant information in the state would make it quite difficult to force representations to be one-to-one, which is the very reason we propose the more relaxed notion of behavioural equivalence.

The usual criteria for correctness of representations corresponds mathematically to homomorphism Our definition of secure refinement uses the machinery of homomorphism in constructing a proof of behavioural equivalence. [Nipkow86] suggests a generalisation of homomorphism called 'simulation' which accommodates non-determinism in observationally equivalent specifications This could be a fruitful line of research for secure refinement, if the restriction to non-deterministic specifications proves to be impractical

In the next section, behavioural equivalence is formally defined as a means to

investigating the soundness of the proof obligation shown in this section.

5 Formal Treatment

5.1 Systems

In this section, Behavioural Equivalence is formally defined, and the proof obligation of the previous section is shown to be valid with respect to that definition.

A state transition relation on a non-empty set of states S, a non-empty set of inputs I, a non-empty set of outputs O, and a non-empty set of operation names OPS defines for each operation, input, and state, possible state, output outcomes. Let the set of all operations OPS, the set of all inputs I, the set of all possible states S and outputs O be fixed.

$$[OPS, I, S, O]$$

The set of all state transition relations is defined as follows:

$$TR == (OPS \times I \times S) \leftrightarrow (S \times O)$$

A system is modelled by a non-empty set of states Σ, a non-empty set of outputs Ω, an initial state α, and a state transition relation *next* on these sets. The set of all systems is defined as follows:

System

$\Sigma : \mathbb{P} S$

$\Omega : \mathbb{P} O$

$\alpha : S$

next : TR

$\alpha \in \Sigma$

$next \in (OPS \times I \times \Sigma) \leftrightarrow (\Sigma \times \Omega)$

$dom\ next = (OPS \times I \times \Sigma)$

The initial state is a state of the system, the state transition relation is restricted to the states and outputs of the system, and the transition relation is total.

5.2 Behaviour

A system scenario, is a sequence of operations invoked with their inputs, and the corresponding output "behaviour" is a sequence of states and outputs The set of all possible input scenarios is called *Scene*, and the set of all possible output behaviours is called *Behaviour*.

$$Scene == seq\ (OPS \times I)$$
$$Behaviour == seq\ (S \times O)$$

The behaviour of a system with respect to a given initial state is described by the following iterator, which determines for any transition relation *tr*, a function of any starting state *st* - giving the relationship between scenarios and possible behaviours.

$$
\begin{array}{|l}
\hline
\_\,\tilde{*} : TR \rightarrow (S \rightarrow (Scene \leftrightarrow Behaviour)) \\
\hline
\forall\, tr{:}TR;\ st{:}S \bullet \\
\quad (tr\,\tilde{*}\ st)(\{\langle\rangle\}) = \{\langle\rangle\} \ \wedge \\
\quad (\forall\, s{:}Scene \mid s \neq \langle\rangle \bullet \\
\qquad (tr\,\tilde{*}\ st)(\{s\}) = \{p{:}inters \bullet \langle p\rangle \frown (tr\,\tilde{*}\ (fst\ p)(tl\ s))\} \\
\qquad \textit{where} \\
\qquad\quad inters == tr(\{fst(hd\ s), snd(hd\ s),\ st\}) \\
\hline
\end{array}
$$

5.3 Output-Deterministic Systems

We only consider deterministic systems, i.e. any scenario gives rise to a unique output behaviour. The output behaviour corresponding to a behaviour sequence *b:Behaviour* is *b§snd*. This observation is generalised for the iterated state transition relation by the special operator *ssnd*, such that $next\tilde{*}\S ssnd\ \alpha$ is a relation between input scenarios and output behaviours.

$$
\begin{array}{|l}
\hline
[A,B,C] \\
\hline
ssnd : (A \leftrightarrow seq(B \times C)) \rightarrow (A \leftrightarrow seq(C)) \\
\hline
\quad ssnd = (\lambda r{:}(A \leftrightarrow seq(B \times C)) \bullet r \S snds) \\
\textit{where} \\
\quad snds == (\lambda s\ seq(B \times C) \bullet s \S snd) \\
\hline
\end{array}
$$

The set of 'output deterministic' systems, *Dsystem*, is defined as the set of systems with functional input-output behaviours

$$
\begin{array}{|l}
\hline
Dsystem \\
\hline
System \\
\hline
next\tilde{*}\S ssnd \in (\Sigma \rightarrow (Scene \rightarrow seq\ \Omega)) \\
\hline
\end{array}
$$

This property enables us to write the term $next\tilde{*}\S ssnd\ \alpha\ s$ to refer unambiguously to the output behaviour of a deterministic system started in its initial state α, executing the input scenario s

Lemma L1. An output deterministic state transition function, iterated on an empty scenario, yields an empty output behaviour

$$Dsystem \vdash (next\tilde{*}\S ssnd)\ \alpha\ \langle\rangle = \langle\rangle$$

5.4 Behavioural Equivalence

Two systems, A and B, with the same input set I and operation names OPS are behaviourally equivalent, written $A\ be\ B$, if and only if both systems produce equivalent behaviours for any conceivable scenario s, up to a bijective interpretation of outputs *interp*

$$
\begin{array}{|l}
\hline
\_be\_ : \mathrm{Dsystem} \leftrightarrow \mathrm{Dsystem} \\
\hline
\forall a,b{:}\mathrm{Dsystem} \bullet \\
\quad a \,be\, b \Leftrightarrow \\
\quad\quad (a \mapsto b) \in \{\, \mathrm{Dsystem}_A; \mathrm{Dsystem}_B \mid \\
\quad\quad\quad\quad (\exists interp{:}(\mathrm{OPS} \times \Omega_B) \rightarrowtail \Omega_A \bullet \forall s{:}\mathrm{Scene} \bullet \\
\quad\quad\quad\quad\quad (next_A \circledS snd) \; \alpha_A\, s = (s \S fst \chi ((next_B \circledS snd) \; \alpha_B\, s) \S interp))\}
\end{array}
$$

where the operator χ combines two equal length sequences in a pointwise fashion, thus:

$$
\begin{array}{|l}
\hline
[A,B] \\
\hline
\_\chi\_ : (\mathrm{seq}(A) \times \mathrm{seq}(B)) \rightarrowtail \mathrm{seq}(A \times B) \\
\hline
\_\chi\_ = \lambda a{:}\mathrm{seq}(A); b{:}\mathrm{seq}(B) \mid \#a = \#b \bullet \lambda i{:}\mathrm{dom}\, a \bullet (a\, i, b\, i) \\
\hline
\end{array}
$$

Lemma L2: The pointwise combination of the empty sequence is the empty sequence:

$$\vdash \langle\rangle \chi \langle\rangle = \langle\rangle$$

5.5 Proof Obligations

The set of all possible outputs from a given operation *op*, written *o op tr* is defined as follows:

$$
\begin{array}{|l}
\hline
o : \mathrm{OPS} \rightarrow \mathrm{TR} \rightarrow \mathbb{F} O \\
\hline
o = \lambda op{:}\mathrm{OPS} \bullet \lambda\, tr{:}TR \bullet \mathrm{ran}(\mathrm{ran}((\{op\} \times I \times S) \lhd tr) \\
\hline
\end{array}
$$

Proposition Two output deterministic systems, A and B, are (*maybe*) behaviourally equivalent with respect to an abstraction relation *abs* if each operation induces a total bijection on the outputs of the corresponding state transition functions, thus

$$
\begin{array}{|l}
\hline
\mathrm{maybe} \;\; (S \leftrightarrow S) \rightarrow (\mathrm{Dsystem} \leftrightarrow \mathrm{Dsystem}) \\
\hline
\mathrm{maybe} = \lambda \mathrm{abs}\; S \leftrightarrow S \bullet \\
\quad\quad \{\, \mathrm{Dsystem}_A, \mathrm{Dsystem}_B \mid \\
\quad\quad\quad \mathrm{abs} \in \Sigma_B \leftrightarrow \Sigma_A \wedge (\sigma_B \rightarrowtail \alpha_A) \in \mathrm{abs} \wedge \\
\quad\quad\quad \forall op{:}\mathrm{OPS} \bullet \\
\quad\quad\quad\quad \mathrm{outabs} \in (o\; op\; next_B) \rightarrowtail (o\; op\; next_A) \\
\quad\quad\quad\quad \textit{where} \\
\quad\quad\quad\quad\quad \mathrm{outabs} == \{\, i{:}I, \sigma_A{:}\Sigma_A; \sigma_B{:}\Sigma_B \mid \sigma_B \mapsto \sigma_A \in \mathrm{abs} \bullet \\
\quad\quad\quad\quad\quad\quad\quad (next_A \S snd\, (op, i, \sigma_A), next_B \S snd\, (op, i, \sigma_B))\}
\end{array}
$$

This is not an alternative definition of behavioural equivalence, but is a sufficient condition on the operations of a system to guarantee it

5.6 Validity

Refinement can be summarised by the following collective proof obligations: the specifications must be *output deterministic*; One specification must *safety-refine* the other - this process yields an abstraction relation and a proof that all operations are monotonic with respect to this; and finally a proof that corresponding operations from the two specifications each have a bijective output correspondence. These proof obligations are specified in the following schema:

Proof_oblig

$abs : S \leftrightarrow S$

$Dsystem_A ; Dsystem_B$

$abs \in \Sigma_B \leftrightarrow \Sigma_A$

$(\sigma_B \mapsto \sigma_A) \in abs$

$\forall op : OPS ; i : \mathbb{I} ; \sigma_A : \Sigma_A ; \sigma_B : \Sigma_B \mid \sigma_B \mapsto \sigma_A \in abs \bullet$
$\quad abs\,(next_B\,fst!((op,i,\sigma_B))) \ni next_A\,fst!((op,i,\sigma_A))$

$(\theta Dsystem_A, \theta Dsystem_B) \in (may\,be\,abs)$

The validity of *maybe* as a proof obligation is stated as follows.

Proof_oblig

$\vdash$

$\theta Dsystem_A$ be $\theta Dsystem_B$

5.7 Validity Proof Outline

An outline proof of validity for the proof obligation follows Examining the definition of be, we show for every pair of deterministic systems, the existence of a bijective interpretation function which translates all outputs from one system to the corresponding outputs of the other There are four main steps:

1. Generalisation prove validity for any two systems A and B
2. Existence posit an assignment for *interp*
3. Bijectivity: prove that *interp* is bijective
4. Induction prove that *interp* translates all output sequences from B as corresponding output sequences from A.

Step 1 simply involves dropping the universal quantifier, reducing the argument to any two systems $Dsystem_A$ and $Dsystem_B$ Step 2 constructs the following assignment for *interp*, based on *absout* in the definition of *maybe*·

$$interp = \bigcup \{ op : OPS \bullet \{ i : \mathbb{I} ; \sigma_A : \Sigma_A ; \sigma_B : \Sigma_B \mid \sigma_B \mapsto \sigma_A \in abs \bullet (op, next_A\,snd\,(op, i, \sigma_A)), next_B\,snd\,(op, i, \sigma_B)) \}$$

Step 3 shows that *interp* is bijective, i e

$$interp \in (OPS \times \Omega_B) \rightarrowtail \Omega_A$$

The antecedent of the proof obligation establishes for each operation *op* that the function *obsout*, i.e.

$$[i:I; \sigma_A:\Sigma_A; \sigma_B:\Sigma_B \mid \sigma_B \mapsto \sigma_A \in abs \cdot (next_A\S snd(op, i, \sigma_A), next_B\S snd(op, i, \sigma_B))]$$

is a total bijection on the inputs and outputs of the corresponding operation. Our assignment to *interp* is the union of all of these bijections, each labelled with its *op*. This labelling partitions the bijections, so the union is also bijective. The union is total, since each component bijection is total, and there is one component for every operation.

The main proof step (4) is requires us to prove that *interp* actually translates the outputs of system *B* to the outputs of system *A*, i.e.

$$\forall s:Scene \cdot (next_A{}^*\S ssnd)\ \alpha_A\, s = (s\S fst \chi\ ((next_B{}^*\S ssnd)\ \alpha_B\, s)\S interp)$$

which is proven by induction on the sequence *s*.

The base of the induction, with *s*=<> is as follows:

$$(next_A{}^*\S ssnd)\ \alpha_A <> = (<>\S fst \chi\ ((next_B{}^*\S ssnd)\ \alpha_B <>)\S interp)$$

Applying lemma L1 to the left and right of the base case, reduces it as follows

$$<> = (<>\S fst \chi\ <>)\S interp$$

which is discharged by lemma L2, since the composition of an empty sequence with a function is the empty sequence

The induction step is as follows Assuming the property holds for a non-empty scenario *t*, i e

$$(next_A{}^*\S ssnd)\ \alpha_A\, t = (t\S fst \chi\ ((next_B{}^*\S ssnd)\ \alpha_B\, t)\S interp)$$

prove it holds for the scenario *t* extended with the operation *op* and input *i*, i e *s*=*t*$^\frown$<(*op*,*i*)>, as follows.

$$(next_A{}^*\S ssnd)\ \alpha_A\, (t^\frown<(op,i)>) =$$
$$((t^\frown<(op,i)>)\S fst \chi\ ((next_B{}^*\S ssnd)\ \alpha_B\, (t^\frown<(op,i)>))\S interp)$$

This is the case if the output behaviours of *A* and *B* correspond over the initial scenario *t*, as well as the additional operation, *op* with input *i* (which amounts to distributing the iterated transition functions over sequence catenation, and further distributing *interp* through sequence catenation). The corresponding subgoals are

(i) $(next_A{}^*\S ssnd)\ \alpha_A\, t = (t\S fst \chi\ ((next_B{}^*\S ssnd)\ \alpha_B\, t)\S interp$

(ii) $\forall \sigma_A : snd(last(next_A{}^*\ \alpha_A\, t)),\ \sigma_B : snd(last(next_B{}^*\ \alpha_B\, t)) \cdot$
$next_A\S snd(op,i,\sigma_A) = inter(op, next_B\S snd(op,i,\sigma_B))$

The first subgoal (i) is the induction hypothesis The second subgoal (ii) requires us to show for the extra input, that the interpreted output from B is identical to the output from A, for each possible state σ_A after executing *A* over *t*, and each corresponding state σ_B after executing *B* over *t* The proof step is valid because the systems are deterministic, and the states σ_A and σ_B are guaranteed to be 'equivalent' up to *abs* (safety refinement)

In other words, both systems make the extra state-transition from the same starting point. The proof of subgoal (ii) follows by generalisation on σ_A and σ_E, and expanding the definition of *inter*

This outline proof has been mechanically checked using the B-tool [BP1]. The only additional simplification made was to exclude "type-checking" from the proof lemmas. It would be valuable to fully elaborate the formal proof, and to remove simplifying assumptions such as unique initial states. Also, the relationship with other refinement activities should be investigated, perhaps integrating the process of constructing output abstractions into the safety refinement process

6 Conclusions

As the specification of a multi-level secure system is refined, extraneous behaviours introduce signalling channels Isomorphic refinements do not expose any more signalling channels than are already present in the specification, but our experience shows that eliminating implementation tricks such as redundancy and unreachable states, increases the mathematical complexity of refinement. Using a looser notion of secure refinement based on behavioural equivalence, in concert with a data hiding mechanism, the implementer is afforded greater freedom without any loss of information flow security We have shown that it is easy to prove behavioural equivalence of deterministic systems For non-deterministic systems, it might be possible to define an analogous notion based on 'simulation'.

We have attempted to characterise secure refinement as liberally as possible, and proposed suitable proof obligations, but as our definition is independent of the security policy, it is slightly over-cautious, although not as over-cautious as isomorphism However, the process of discharging the new proof obligation is preferable to re-addressing the complex security policy correspondence at each refinement step

7 Acknowledgements

This work has been carried out with the support of Procurement Executive Ministry of Defence, under Contract No SLS42c/719, for which Peter Bottomley, Simon Wiseman and Andrew Wood of the Defence Research Agency Electronics Division provided support and guidance

A References

[BP1] Abrial, J-R , "B Reference Manual", BP Research International Ltd 1990

[Goguen78] J A Goguen et al , "The initial algebra approach to the specification, correctness, and implementation of abstract data types", in "Current Trends in Programming Methodology, IV, Data Structuring, pp 80-149, Prentice Hall 1978

[Jones86] C B Jones, 'Systematic Software Development Using VDM , Prentice Hall, 1986

[Nipkow86] Nipkow, T , Non-deterministic Data Types Models and Implementations , Acta Informatica, 22, 629 661, 1986

[Sannella83] Sannella, D., "Behavioural Equivalence and Algebraic Specification", Workshop on Semantics of Programming Languages, Goteburg, August 1983, pp162-166.

[Sannella84] Sannella, D., et al, "On Observational Equivalence and Algebraic Specification",Univ. of Edinburgh, Draft, July 1984

[Spivey88] Spivey, J M., "Understanding Z A Specification Language and its Formal Semantics", Cambridge University Press, 1988.

[Spivey89] Spivey, J M , "The Z Notation A Reference Manual", Prentice Hall International, 1989.

[Terry89] Terry, P.F., Wiseman, S R , "A 'New' Security Policy Model", Proceedings of the 1989 IEEE Symposium on Security and Privacy, Oakland 1989

[Wiseman90] Wiseman, S , "Basic Mechanisms for Computer Security", RSRE Report 89024, January 1990

REPORT DOCUMENTATION PAGE

DRIC Reference Number (if known) .....

Overall security classification of sheet ..... UNCLASSIFIED .....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the field concerned must be marked to indicate the classification eg (R), (C) or (S).

Originators Reference/Report No.	Month	Year
REPORT 91019	FEBRUARY	1991

Originators Name and Location
RSRE, St Andrews Road
Malvern, Worcs WR14 3PS

Monitoring Agency Name and Location

Title
SECURE REFINEMENT

Report Security Classification	Title Classification (U, R, C or S)
UNCLASSIFIED	U

Foreign Language Title (in the case of translations)

Conference Details

Agency Reference	Contract Number and Period
	SLS42C/719
Project Number	Other References

Authors	Pagination and Ref
SMITH, P	17

Abstract

Behavioural equivalence is identified as a mathematical property which captures the notion of secure refinement

Abstract Classification (U,R,C or S)
U

Descriptors

Distribution Statement (Enter any limitations on the distribution of the document)
UNLIMITED

580/48

INTENTIONALLY BLANK